In [4]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,precision_score, recall_score, roc_auc_score
from sklearn.metrics import f1_score,confusion_matrix, precision_recall_curve, roc_curve
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

data = pd.read_csv('diabetes.csv')
data.head(3)
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0.
1	1	85	66	29	0	26.6	0.
2	8	183	64	0	0	23.3	0.

In [3]:

```
data['Outcome'].value_counts() #Negative가 상대적으로 많다
```

Out[3]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

In [5]:

```
data.info() #Null 값은 없고 feature type은 모두 숫자형이다. feature encoding은 필요 없어 보임
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [16]:

```
# 수정된 get_clf_eval() 함수
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},₩
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))
```

In [17]:

```
def precision_recall_curve_plot(y_test=None, pred_proba_c1=None):
    # threshold ndarray와 이 threshold에 따른 정밀도, 재현율 ndarray 추출.
    precisions, recalls, thresholds = precision_recall_curve( y_test, pred_proba_c1)

    # X축을 threshold값으로, Y축은 정밀도, 재현율 값으로 각각 Plot 수행. 정밀도는 점선으로 표시
    plt.figure(figsize=(8,6))
    threshold_boundary = thresholds.shape[0]
    plt.plot(thresholds, precisions[0:threshold_boundary], linestyle='--', label='precision')
    plt.plot(thresholds, recalls[0:threshold_boundary],label='recall')

    # threshold 값 X 축의 Scale을 0.1 단위로 변경
    start, end = plt.xlim()
    plt.xticks(np.round(np.arange(start, end, 0.1),2))

    # x축, y축 label과 legend, 그리고 grid 설정
    plt.xlabel('Threshold value'); plt.ylabel('Precision and Recall value')
    plt.legend(); plt.grid()
    plt.show()
```

In [15]:

```
#예측 모델 생성
# Outcon이 label 값 = y
y = data['Outcome']
X = data.drop('Outcome',axis =1)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 156, str
atify = y)
# stratify : 한 쪽에 너무 쏠리는 것 방지
```

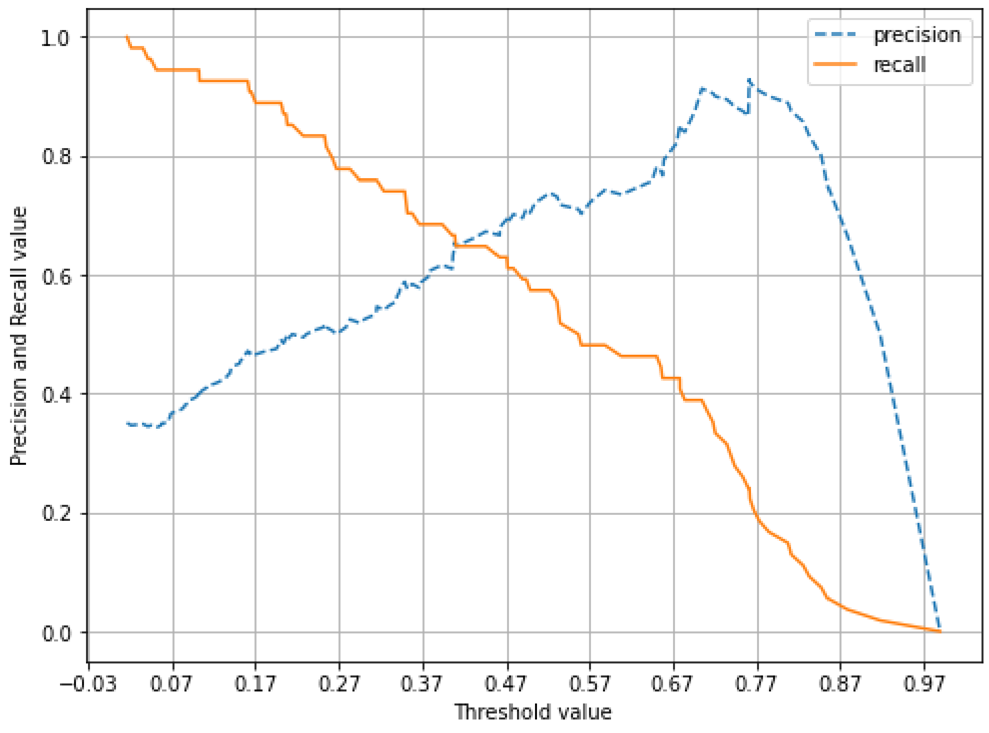
In [19]:

```
#logistic regression fit/predict/evaluate
model = LogisticRegression()
model.fit(X_train,y_train)
pred = model.predict(X_test)
pred_proba = model.predict_proba(X_test)[:,1]

get_clf_eval(y_test,pred,pred_proba)
# 전체 데이터에서 약 65%가 Negative이기 때문에 재현율에 더 focus 맞추기
```

```
오차 행렬
[[88 12]
 [23 31]]
정확도: 0.7727, 정밀도: 0.7209, 재현율: 0.5741,    F1: 0.6392, AUC:0.7919

C:₩Users₩1216k₩.conda₩envs₩test₩lib₩site-packages₩sklearn₩linear_model₩_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [20]:

```
# 임곗값별 정밀도와 재현율 값의 변화
pred_proba_c1 = model.predict_proba(X_test)[:,1]
precision_recall_curve_plot(y_test,pred_proba_c1)

#threshold = 0.42로 하면 두 지표의 균형을 맞출 수 있을 것 같다 -> 하지만 0.6x로 값이 여전히 낮음
-> 피처 값 분포도 확인
```



In [21]:

```
data.describe()

#min = 0인 feature들이 상당히 많음 -> Gulucose(포도당 수치)의 min이 0인 것은 잘못된 데이터
```

Out[21]:

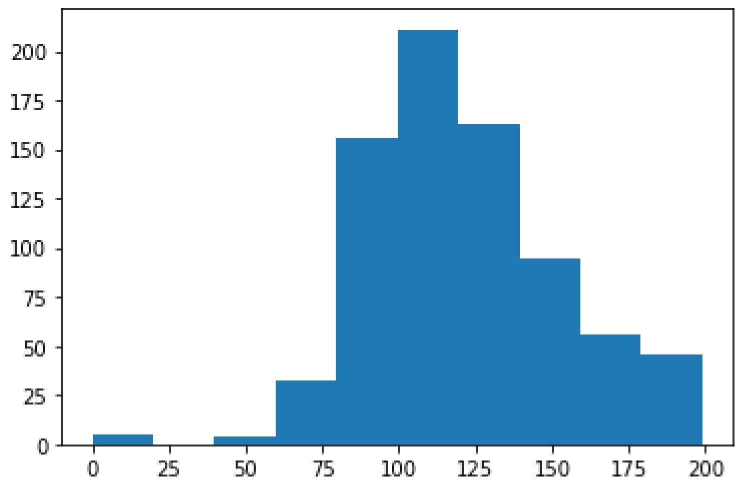
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diat
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [23]:

```
# Glucose 값 분포 확인
plt.hist(data['Glucose'],bins=10)
```

Out[23]:

```
(array([  5.,   0.,   4.,  32., 156., 211., 163.,  95.,  56.,  46.]),
 array([  0. ,  19.9,  39.8,  59.7,  79.6,  99.5, 119.4, 139.3, 159.2,
        179.1, 199. ]),
 <BarContainer object of 10 artists>)
```



In [26]:

```
# min = 0 데이터가 각 피처들에서의 비율 확인
#min = 0인 features
zero_features = ['Glucose','BloodPressure','SkinThickness','Insulin','BMI']

#전체 데이터 건수
total_count = data['Glucose'].count()

#피처별 데이터 값이 0인 건수 추출, 비율 계산
count = 0
for feature in zero_features :
    zero_count = data[data[feature]==0][feature].count() #<-----------------여기 잘 보기!
    print('{0} 0 건수는 {1}, 비율은 {2:.2f} % ₩n'.format(feature,zero_count,100*zero_count/tot
al_count))
# 결과 : SkinThickness, Insulin의 경우 0의 비율이 크기 때문에 일괄적으로 삭제하면 안된다 -> 평균
으로 대체하기!
```

```
Glucose 0 건수는 5, 비율은 0.65 %

BloodPressure 0 건수는 35, 비율은 4.56 %

SkinThickness 0 건수는 227, 비율은 29.56 %

Insulin 0 건수는 374, 비율은 48.70 %

BMI 0 건수는 11, 비율은 1.43 %
```

In [27]:

```
mean_zero_features = data[zero_features].mean()
data[zero_features]=data[zero_features].replace(0,mean_zero_features)
# 일반적으로 로지스틱 회귀의 경우 숫자 데이터에 scaling을 적용하는 것이 good
```

In [28]:

```
#성능 평가 Re
y = data['Outcome']
X= data.iloc[:,:-1]

#StandardScaler class를 이용해서 feature data set에 일괄적으로 스케일링 적용
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.2, random_state = 156, str
atify = y)

#logistic regression fit/predict/evaluate
model = LogisticRegression()
model.fit(X_train,y_train)
pred = model.predict(X_test)
pred_proba = model.predict_proba(X_test)[:,1]

get_clf_eval(y_test,pred,pred_proba)
# 데이터 변환과 스케일링을 통해 성능 수치가 일정 수준 개선됨. but 더 개선이 필요함 -> 임계점 변
화시키기
```

```
오차 행렬
[[89 11]
 [20 34]]
정확도: 0.7987, 정밀도: 0.7556, 재현율: 0.6296,    F1: 0.6869, AUC:0.8413

C:₩Users₩1216k₩.conda₩envs₩test₩lib₩site-packages₩sklearn₩linear_model₩_logistic.p
y:764: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [29]:

```
from sklearn.preprocessing import Binarizer

def get_eval_by_threshold(y_test , pred_proba_c1, thresholds):
    # thresholds 리스트 객체내의 값을 차례로 iteration하면서 Evaluation 수행.
    for custom_threshold in thresholds:
        binarizer = Binarizer(threshold=custom_threshold).fit(pred_proba_c1)
        custom_predict = binarizer.transform(pred_proba_c1)
        print('임곗값:',custom_threshold)
        get_clf_eval(y_test , custom_predict, pred_proba_c1)
```

In [30]:

```
thresholds = [0.3 , 0.33 ,0.36,0.39, 0.42 , 0.45 ,0.48, 0.50]
pred_proba = model.predict_proba(X_test)
get_eval_by_threshold(y_test, pred_proba[:,1].reshape(-1,1), thresholds )
#0.48일 때가 좋아보임
```

```
임곗값: 0.3
오차 행렬
[[69 31]
 [10 44]]
정확도: 0.7338, 정밀도: 0.5867, 재현율: 0.8148,    F1: 0.6822, AUC:0.8413
임곗값: 0.33
오차 행렬
[[73 27]
 [13 41]]
정확도: 0.7403, 정밀도: 0.6029, 재현율: 0.7593,    F1: 0.6721, AUC:0.8413
임곗값: 0.36
오차 행렬
[[77 23]
 [15 39]]
정확도: 0.7532, 정밀도: 0.6290, 재현율: 0.7222,    F1: 0.6724, AUC:0.8413
임곗값: 0.39
오차 행렬
[[77 23]
 [15 39]]
정확도: 0.7532, 정밀도: 0.6290, 재현율: 0.7222,    F1: 0.6724, AUC:0.8413
임곗값: 0.42
오차 행렬
[[80 20]
 [16 38]]
정확도: 0.7662, 정밀도: 0.6552, 재현율: 0.7037,    F1: 0.6786, AUC:0.8413
임곗값: 0.45
오차 행렬
[[82 18]
 [18 36]]
정확도: 0.7662, 정밀도: 0.6667, 재현율: 0.6667,    F1: 0.6667, AUC:0.8413
임곗값: 0.48
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481,    F1: 0.6931, AUC:0.8413
임곗값: 0.5
오차 행렬
[[89 11]
 [20 34]]
정확도: 0.7987, 정밀도: 0.7556, 재현율: 0.6296,    F1: 0.6869, AUC:0.8413
```

In [31]:

```
# 임겟값를 0.48로 설정한 Binarizer 생성
binarizer = Binarizer(threshold=0.48)

# predict_proba() 예측 확률 array에서 1에 해당하는 컬럼값을 Binarizer변환.
pred_th_048 = binarizer.fit_transform(pred_proba[:, 1].reshape(-1,1))

get_clf_eval(y_test , pred_th_048, pred_proba[:, 1])
```

```
오차 행렬
[[88 12]
 [19 35]]
정확도: 0.7987, 정밀도: 0.7447, 재현율: 0.6481,    F1: 0.6931, AUC:0.8413
```